Санкт-Петербургский государственный университет
Прикладная математика и информатика
Вычислительная стохастика и статистические модели

Картышева Елена Николаевна

Улучшение в SPAN модели поиска пиков с помощью модели линейной регрессии

Отчет о преддипломной практике

Научный руководитель:
к. ф.-м. н., доцент Н. П. Алексеева

Санкт-Петербург
2019

Оглавление

Введение

Организм любого живого существа состоит из клеток. Зачастую клетки одного организма разительно отличаются друг от друга (например, клетки волоса совершенно не похожи на клетки кожи или нейроны), хотя все они (за исключением половых) несут один и тот же набор наследственной информации. Эти различия объясняются многими факторами, в том числе изменением уровня экспрессии определенных генов, которые связаны с активностью того или иного регуляторного белка. По этим причинам важно понимать, как изменятся связывание белков с ДНК.

Для изучения связывания используется метод ChIP-seq, который, к сожалению, является довольно неточным, поэтому обработка выходных данных сильно влияет на результаты дальнейших исследований. На данный момент существует несколько программ, которые обрабатывают ChIP-seq данные с помощью математических моделей. В данной работе изучается программа SPAN и возможности ее улучшения.

1. Описание задачи

Целью работы было изучить математическую модель, используемую SPAN и возможности ее улучшения. Очевидными требованиями является корректная работа программы и достаточное быстрое время работы.

Задачи, поставленные для достижения цели:

1. Изучить модель использующуюся в SPAN (раздел 2).
2. Изучить уже существующие аналоги и подходы к решению задач.
3. Реализация обобщенной линейной модели в Kotlin/Java (раздел 5) с дальнейшей возможностью внедрения в проект.

2. Некоторые биологические сведения

Геном — это совокупность всей наследственной информации организма. Эта информация разделена на крупные блоки — хромосомы. Каждая хромосома — это отдельная молекула ДНК. Молекула ДНК состоит из двух линейных полимеров, которые в свою

очередь представляют собой последовательность четырех нуклеотидов: A, C, G, T. Полимеры располагаются друг напротив друга так, что нуклеотиды одной цепи, образуют пары с нуклеотидами другой цепи. Размер геномов обычно определяется количеством таких пар. Число пар нуклеотидов равно числу нуклеотидов в одной из цепочек. Термин «пара нуклеотидов» обычно сокращается до bp (base pair).

ChIP-seq — метод анализа ДНК-белковых взаимодействий, для поиска мест связывания транскрипционных факторов и других исследований генома.

Сайт связывания — особый участок в целой молекуле, который способен вступать в стабильное взаимодействие с другими молекулами.

В качестве выходных данных ChIP-seq получаем набор **треков** — пары хромосома – позиция на ней.

В местах связывания треков будет значительно больше, то есть будут образовываться пики. Однако у ChIP-seq метода есть проблема: треки появляются и в местах, где связывания нет. За счет этого на данных появляется шум. Таким образом, чтобы находить сайты связывания, необходимо научиться отделять пики от шумов. Для этого существуют различные программы, одна из них SPAN [1].

3. Модель поиска пиков в SPAN

SPAN (Semi-supervised Peak ANalyzer) — инструмент для поиска областей повышенного сигнала в геномных данных ChIP-seq, разработанный лабораторией BioLabs компании JetBrains.

Каждая хромосома разделяется на бины (стандартный размер бина - 200bp, он может быть изменен пользователем). Для каждого бина считается покрытие — количество треков, которые попали в этот бин.

Поиск пиков осуществляется с помощью скрытой марковской модели (HMM) с тремя скрытыми состояниями:

1. NULL (для участков хромосомы, на которую треки не попали)
2. LOW и HIGH (два негативно-биномиальных с различными параметрами для пиков и шумов).

Формализуем постановку задачи: имея последовательность открытых состояний (покрытие), хотим найти наиболее вероятную последовательность скрытых состояний

(NULL, LOW или HIGH).

- $O_1, \ldots, O_T$ — последовательность открытых состояний,
- $\Pi = \pi_i$ — начальные вероятности,
- $A = a_{ij}$ — матрица переходов,
- $B = \{b_i(O)\}$ — матрица вероятности генерации скрытых состояний.

Задача: при известной последовательности $O = O_1, \ldots, O_T$ открытых состояний найти Π, A, B, максимизируя вероятность наблюдений O.

3.1. Алгоритм Баума-Велча

1. Инициализируем Π, A, B случайно
2. **Прямой ход**

   Пусть

   $$\alpha_i(t) = P(O_1, \ldots, O_t, S_t = j | \Pi, A, B)$$

   — вероятность генерации последовательности открытых состояний $O_1, \ldots, O_t$ и при этом в момент t модель находится в скрытом состоянии j.

   $\alpha_j(t)$ вычисляются итеративно:

   $$\alpha_j(t) = \sum_i \alpha_i(t-1) a_{ij} b_j(O_{t-1}),\ \alpha_j(1) = \pi_j b_j(O_1).$$

   Тогда вероятность генерации последовательности определяется следующим образом:

   $$P(O_1, \ldots, O_T) = \sum_i \alpha_i(T).$$

3. **Обратный ход**

   Пусть

   $$\beta_i(t) = P(O_{t+1}, \ldots, O_T | S_t = j)$$

   — суммарная вероятность генерации последовательности открытых состояний $O_{t+1}, \ldots, O_T$ при условии того, что в момент t модель находилась в скрытом состоянии i.

Вычисляем $\beta_i(t)$ итеративно:

$$\beta_i(t) = \sum_j \beta_j(t+1) a_{ij} b_j(O_{t+1}),\ \beta_i(T) = 1.$$

4. **Пересчет коэффициентов**

Используя $\alpha_j(t)$ и $\beta_i(t)$ вычисляем вероятности перехода из состояния i в состояние j в момент t:

$$p_{ij}(t) = \frac{\alpha_i(t) a_{ij} b_j(O_{t+1}) \beta_j(t+1)}{\sum_m \alpha(T)}.$$

1 Также вычисляем вероятности оказаться в состоянии i в момент t:

$$\gamma_i(t) = \sum_j p_{ij}(t).$$

Тогда коэффициенты пересчитываются следующим образом:

$$\hat{\pi}_i = \gamma_i(1),$$
$$\hat{a}_{ij} = \frac{\sum_t p_{ij}(t)}{\sum_t \gamma_i(t)},$$
$$\hat{b}_i(O_k) = \frac{\sum_{t, O_t = O_k} \gamma_i(t)}{\sum_t \gamma_i(t)}.$$

Шаги 2-4 продолжаем до тех пока, пока коэффициенты не перестанут заметно изменяться.

4. Добавление ковариатов к модели

Из всех программ, которые производят обработку ChIP-seq данных, наиболее близкой по используемой модели является ZINBA [2].

Модель поиска пиков в ZINBA использует смесь, состоящую из двух отрицательно-биномиальных распределений (с различными параметрами для шумов и пиков) и распределения Дирака, сосредоточенное в нуле (для мест, где сигнал отсутствует). Модель ZINBA позволяет использовать дополнительные характеристики о геноме, чтобы улучшить предсказание. Для добавления дополнительных парамеров используются обощенные линейные модели [3].

4.1. Обобщенные линейные модели

Рассмотрим обычную модель линейной регрессии. Пусть $b \in \mathbb{R}^m,\ b \neq 0$ — вектор отклика, $A \in \mathbb{R}^{m \times n},\ m \geq n,\ rk(A) = n$ — матрица плана. Тогда линейная модель описывается:

$$Ax = b + e,$$

где $x \in \mathbb{R}^m$ — коэффициенты модели, а $e \in \mathbb{R}^m$ — вектор случайных ошибок (независимые, $\mathbb{M}(e) = 0$, $\mathbb{D}(e) = const$).

Теперь позволим некоторые послабления (автокорреляция e, изменение дисперсии e, нелинейная зависимость b от Ax), иными словами «обобщаем» линейную модель.

$$g(Ax) = b + e, \quad Var(e) = \sigma^2 W^{-1},$$

где $g(x)$ — линк-функция, $\sigma^2 W^{-1}$ — матрица ковариаций e, W — положительно-определенная матрица, σ

На практике W обычно неизвестно и зависит от распределения ошибок. Пусть ϵ независимые случайные величины с различной диперсией. Тогда матрица W будет диагональной.

Решение может быть найдено методом наименьших квадратов:

$$\min_x (g(Ax) - b)^T W (g(Ax) - b).$$

Численное решение находится с помощью алгоритма IRLS (Iteratively Reweighted

Least Squares estimation)

Алгоритм 1: IRLS

Исходные параметры: Матрица плана A,

Вектор наблюдений b,

Линк-функция g,

Производная линк функции g',

Функция дисперсии var,

Максимальное количество итераций itmax

Точность tol

Результат: Коэффициенты модели x_{j+1}

Пусть $x_1 = 0$;

цикл $j = 1,2, \ldots, itmax$ **выполнять**

  $\nu = Ax_j$;

  $z = \nu + \frac{b-g(\nu)}{g'(\nu)}$;

  $W = diag\left(\frac{g'(\nu)^2}{var(g(\nu))}\right)$;

  $x_{j+1} = (A^T W A)^{-1} A^T W z$;

  если $\|x_{j+1} - x_j\| < tol$ **тогда**

    Stop

  конец

конец

4.2. Применение обощенных линейных моделей к SPAN

В SPAN обобщенная линейная модель используется, чтобы ввести дополнительные ковариаты к модели.

Изначально модель имеет следующие параметры:

- Веса компонент смеси
- Среднее (μ_1, μ_2) и количество неудач (f_1, f_2) для шумов и пиков соответственно.

Входные данные алгоритма:

- Вектор $n \times 1$ $Y = (Y_1, \ldots, Y_n)$, где Y_i — покрытие i-го бина.
- Матрица $n \times (p+1)$ X_1 — матрица ковариат для компоненты с шумом.

- Матрица $n \times (q+1)$ X_2 — матрица ковариат для компоненты с пиками. p, q — количество ковариат для компоненты с шумом и пиками соответственно.

Параметры скрытой марковской модели обучаются с помощью EM-алгоритма. Чтобы добавить ковариаты к модели, некоторые параметры будут предсказываться не напрямую, а с помощью обобщенных линейных моделей:

$\log(\mu_1) = X_1\beta_1$, $\log(\mu_2) = X_2\beta_2$.

5. Взвешенная линейная регрессия на Java

Взвешенная регрессия реализована для Java в библиотеке Apache Commons Math.

Класс `GLSMultipleLinearRegression` ищет параметры регрессии, используя в качестве данных матрицу данных X (формат `double[][]`), вектор наблюдений y (формат `double[]`) и матрицу ковариций Ω (формат `double[][]`). При создании Ω не учитывается её диагональность, т.е. диагональная матрица хранится не сжато, а как полный двумерный массив.

Здесь возникает проблема: размер матрицы Ω — $N \times N$, где N — количество бинов на хромосоме. Для примера рассмотрим первую хромосому человека. Её длина приблизительно 250 bp. Если взять стандартный бин длиной 200bp, получим 1.25 млн бинов, а это $1.5 * 10^{12}$ элементов матрицы. Один объект типа double занимает 8 б, соответственно вся матрица займет $1.5 * 10^{12} * 8$ б $= 12$ террабайт оперативной памяти. Для хранения соответствующего вектора весов потребуется лишь $1.25 * 10^6 * 8$ б $= 10$ Мб.

К счастью, для алгоритма IRLS нужен лишь вектор весов, а не матрица, поэтому замена `double[][]` covariance в методе newSampleData на `double[]` variance сэкономит много памяти.

Для хранения матрицы X используется интерфейс `RealMatrix`, для хранения Ω — класс `DiagonalMatrix`, имплементация `RealMatrix`. `RealMatrix` имеет метод `multiply`, который в качестве аргумента принимает значение типа `RealMatrix`, поэтому при вызове

```
1                RealMatrix XTOTX = XT.multiply(Omega).multiply(this.getX());
```

матрица Omega приведется к `RealMatrix` и станет занимать много памяти. Метод `multiply` в `DiagonalMatrix` Поэтому необходимо проследить за тем, чтобы матричные

умножения везде выполнялись эффективно.

Реализация IRLS на Kotlin

Так как Kotlin полностью совместим с Java, то в нем можно использовать любые библиотеки, написанные на Java, что дает возможность использовать класс взвешенной регресии в проекте, написанном на Kotlin.

```
1         //df - DataFrame with model matrix and response vector
2         //t - number of column where response vector contains
3
4         fun IRLS(df: DataFrame, t:Int){
5                 var y = df.sliceAsDouble(df.labels[t].intern())
6                 var xFromCovariates = emptyArray<DoubleArray>()
7                 for (label in covariateLabels){
8                         xFromCovariates += df.sliceAsDouble(label)
9                 }
10
11                var x  = Array2DRowRealMatrix(xFromCovariates).transpose().data
12
13                var wlr = WLSMultipleLinearRegression()
14                val weights_0 = DoubleArray(x.size)
15                Arrays.fill(weights_0, 0.0)
16                wlr.newSampleData(y, x, weights_0)
17
18                val iterMax = 250
19                val tol = 1e-8
20
21                //fill x0 with zeros
22                val x0 = DoubleArray(x[0].size+1)
23                Arrays.fill(x0, 0.0)
24
25                var X0: RealVector = ArrayRealVector(x0)
26                var X1:RealVector = ArrayRealVector(x0)
27
28                val X = wlr.getx()
29                val Y = ArrayRealVector(y)
30
31                for (i in 0 until iterMax) {
32                        val eta = X.operate(X0)
33                        val countedLink = eta.map { link(it) }
34                        val countedLinkDeriv = eta.map { linkDerivative(it) }
35                        val z:RealVector = eta
36                                .add(Y.subtract(countedLink)
37                                .ebeDivide(countedLinkDeriv))
38                        val countedLinkVar = countedLink.map { linkVariance(it) }
39                        val W = countedLinkDeriv
40                                .ebeMultiply(countedLinkDeriv)
```

```
                                .ebeDivide(countedLinkVar).toArray()

                        wlr.newSampleData(z.toArray(), x, W)

                        X1 = wlr.calculateBeta()
                        if (X1.subtract(X0).l1Norm < tol) {
                                println("Iterations completed: ${i}")
                                break
                        }
                        X0 = X1
                }
                this.regressionCoefficients = X1.toArray()
        }
```

Проверка работы взвешенной регрессии:

Полученный вектор параметров:

$[13.68203, -0.84101, -0.52535, 0.07949, 0.26359, 0.40841]$.

Вектор параметров, полученный функцией lm в R:

$[13.68203, -0.84101, -0.52535, 0.07949, 0.26359, 0.40841]$.

Проверка работы алгоритма IRLS:

В качестве примера рассматривалась модель пуассоновской регрессии с линк-фукнцией $g(x) = e^x$, поскольку именно пуассоновскую регрессию и планируется в дальнейшем применять к модели.

Проверка проводилась следующим образом: матрица плана размером $10^6 \times 3$ заполнялась случайными числами из отрезка $(0, 1)$. Используя вектор параметров $(0, 4, -2)$ генерировался вектор наблюдений. Используя сгенерированный вектор наблюдений и матрицу плана 100 раз запускался алгоритм IRLS.

Максимальное отличие между истинным вектором параметров и предсказанным: $[0.0049, 0.0057, 0.0034]$.

Среднее время работы алгоритма: 7.5 sec

Количество итераций: 19

6. Результаты

В ходе работы были изучены:

- модель поиска пиков в SPAN.

- обобщенная линейная модель и ее применение в ZINBA.

Практические результаты:

- Дополнен класс `GLSMultipleLinearRegression` в Apache Commons, позволяющий использовать данный класс на больших данных и создан pull request с обновлениями.
- Реализован класс взвешенной регрессии в Java для использования до релиза Apache Commons.
- Класс взвешенной регрессии внедрен в проект и готов к использованию.

Список литературы

1. SPAN Peak Analyzer. — Access mode: https://research.jetbrains.org/groups/biolabs/tools/span-peak-analyzer.
2. ZINBA integrates local covariates with DNA-seq data to identify broad and narrow regions of enrichment even within amplified genomic regions / Naim U. Rashid, Paul G. Giresi, Joseph G. Ibrahim et al. // Genome Biology. — 2011.
3. Rashid Naim U., Giresi Paul G., Ibrahim Joseph G. et al. ZINBA supplementary methods. — 2011.